

OFFENLEGUNG

Abweichungen von den offiziellen VDA5050-Schemata

Stand 14. August 2026 · Für Anwender, Prüfhäuser und das VDA5050-Gremium

LogIQ VDA5050 prüft aufgezeichneten Verkehr gegen die veröffentlichten JSON-Schemata aus github.com/VDA5050/VDA5050. An zehn Stellen können wir diese Dateien nicht so verwenden, wie sie dort liegen. Dieses Dokument nennt jede einzelne, mit Quelle, Belegstelle und Begründung.

Wir führen diese Abweichungen nicht, weil wir den Standard für verbesserungswürdig halten. Wir führen sie, weil eine Datei sonst nicht lesbar wäre oder weil sie dem normativen Fließtext derselben Spezifikation widerspricht.

Die Regel, der wir folgen

Die Spezifikation regelt den Konfliktfall selbst:

„If there are differences between the JSON schemas and this document, the variant in this document applies.“

– VDA5050 2.1.0, Abschnitt 6 · VDA5050 3.0.0, Abschnitt 7

Widersprechen sich Schema und Fließtext, gilt der Fließtext. Genau das tun wir. Jede inhaltliche Abweichung unten folgt dem Dokument gegen sein eigenes Schema. Keine folgt unserer Auffassung gegen beide.

Prüfstand

Geprüft wurde gegen den **Kopf des Release-Branches**, nicht gegen das Tag: die Branches tragen die Korrekturen, die nach einem Release entstehen. [release/2.0.0](#) allein liegt 22 Commits über seinem Tag.

Version	Branch	Commit	Commit-Datum
1.1	<code>release/1.1.0</code> (nie getaggt)	<code>5457824</code>	02.06.2023
2.0	<code>release/2.0.0</code>	<code>33d7dc2</code>	20.09.2024
2.1	<code>release/2.1.0</code>	<code>64788b6</code>	19.08.2024
3.0	<code>release/3.0.0</code>	<code>baf900a</code>	19.03.2026

Abgerufen am 14.08.2026. Sämtliche Aussagen dieses Dokuments wurden an diesem Tag gegen diese Commits nachgeprüft.

Klasse A: die Datei ist kein gültiges JSON

Vier veröffentlichte Schemata enthalten ein Komma vor der schließenden Klammer. Ein strikter Parser lehnt sie ab; sie sind ohne Eingriff nicht einlesbar.

Datei	Version	Zeile	Fehler
<code>state.schema</code>	2.0	220	Komma vor Array-Ende
<code>factsheet.schema</code>	3.0	173	Komma vor Objektende
<code>order.schema</code>	3.0	314	Komma vor Objektende
<code>visualization.schema</code>	3.0	54	Komma vor Objektende

Was wir tun: Wir entfernen das überzählige Komma und lesen die Datei sonst unverändert ein. Der Inhalt ändert sich dadurch nicht. Ein Komma vor der schließenden Klammer trägt keine Bedeutung.

Klasse B: das Schema widerspricht dem Dokument

Vier Stellen, an denen das Schema etwas anderes verlangt als der Fließtext derselben Version. Wir folgen dem Fließtext.

`blockingTypes` ist nicht erfüllbar (Factsheet 2.0 und 2.1)

Das Schema definiert:

```
"blockingTypes": {
  "type": "array",
  "description": "Array of possible blocking types for defined action.",
  "enum": ["NONE", "SOFT", "HARD"]
}
```

Der Wert muss ein Array sein **und** gleichzeitig einer der drei Zeichenketten entsprechen. Beides zusammen kann kein Wert erfüllen. In dieser Fassung besteht kein einziges Factsheet die Prüfung, auch kein vollständig korrektes.

Was wir tun: Wir verschieben das `enum` unter `items`, wie es die eigene Beschreibung des Feldes verlangt. VDA5050 3.0 schreibt es selbst so und ergänzt dort zusätzlich den Wert `SINGLE`.

`mobileRobotKinematic` ist nirgends definiert (Factsheet 3.0)

`typeSpecification.required` verlangt `mobileRobotKinematic`. Dieses Feld definiert das Schema an keiner Stelle. Definiert ist `mobileRobotKinematics`, mit s und genau so nennt es auch Abschnitt 7.10. Die Schreibweise ohne s kommt im gesamten Spezifikationsdokument kein einziges Mal vor.

Was wir tun: Wir setzen `required` auf den Namen, den Schema und Fließtext tatsächlich führen.

`pauseAllowed` und `cancelAllowed` haben den falschen Typ (Factsheet 3.0)

Beide Felder sind im Schema als `string` deklariert. Die Feldtabelle in Abschnitt 7.10 führt für beide den Typ `boolean` und ihre Beschreibungen beginnen mit `"true":` und `"false":`.

Was wir tun: Wir typisieren beide als `boolean`.

`requestStatus` kennt `QUEUED` nicht (State 3.0)

Abschnitt 6.9 ist eindeutig:

„The field `requestStatus` describes the life cycle of the request and shall support the following values: [...] 'QUEUED': Acknowledge the mobile robot's request to the fleet control, but no permission is given yet.“

Das Schema führt für `zoneRequest.requestStatus` und `edgeRequest.requestStatus` nur vier Werte: `REQUESTED`, `GRANTED`, `REVOKED`, `EXPIRED`. Ein Fahrzeug, das eine eingereihte Anfrage meldet, hat damit keinen zulässigen Status dafür. Dabei darf die Flottensteuerung laut demselben Abschnitt genau mit `QUEUED` antworten.

Anzumerken ist: Auch die Feldtabelle desselben Dokuments führt nur die vier Werte. Der Widerspruch liegt damit innerhalb des Dokuments. Wir folgen dem normativen Satz mit „shall support“, nicht der Tabelle.

Was wir tun: Wir ergänzen `QUEUED` in beiden Enums.

Die Zonenfaktoren tragen ein Leerzeichen am Namensende (zoneSet 3.0)

Die Zweige `PRIORITY` und `PENALTY` definieren ihren Faktor als `priorityFactor` und `penaltyFactor`, mit einem Leerzeichen am Ende des Namens, während `required` in derselben Datei beide ohne Leerzeichen nennt.

Abgelehnt wird dadurch nichts, geprüft aber auch nichts. Die Grenzen, die die Norm nennt, 0,0 bis 1,0, liegen auf einer Eigenschaft, die keine Nachricht trägt; ein `priorityFactor` von 7,5 kommt unbemerkt durch. Ein Schema, das alles durchlässt, ist genau der Fehler, den ein Konformitätswerkzeug nicht erben darf.

Was wir tun: Wir entfernen das Leerzeichen aus beiden Namen. Erst dadurch gilt der von der Norm genannte Bereich für das Feld, das ein Fahrzeug tatsächlich sendet.

`PAUSED` fehlt unter den Aktionsstatus (state 2.1.0)

Die State-Tabelle führt sechs Werte und verweist auf den Abschnitt, der sie beschreibt:

Enum {'WAITING', 'INITIALIZING', 'RUNNING', 'PAUSED', 'FINISHED', 'FAILED'} See Section 6.11
actionStates.

Abschnitt 6.11 beschreibt `'PAUSED'` als „the action is paused because of a pause instantAction or external trigger (pause button on the AGV)“. Das Enum des Schemas trägt fünf, und seine eigene `description` desselben Feldes nennt `PAUSED` weiterhin. Der Wert wurde aus der Aufzählung genommen und im Satz daneben stehen gelassen.

2.0.0 hat alle sechs. Mit `startPause` hält eine Leitsteuerung ein Fahrzeug an einer Kreuzung an, und ein pausiertes Fahrzeug hat sonst keinen Status, mit dem es die Pause melden könnte: `RUNNING` ist unwahr, `WAITING` heißt „wartet auf einen Auslöser“.

Was wir tun: `PAUSED` ins Enum aufnehmen.

Klasse C: eine Version ohne Schemata

VDA5050 1.1 ist älter als der Ordner `json_schemas/` und veröffentlicht keine Schemata. Der Ordner existiert im Branch nicht.

Was wir tun: Wir leiten die 1.1-Schemata aus 2.1.0 ab. Pflichtfelder, Typen und Enums kommen von dort, ohne das Factsheet-Topic, das 1.1 nicht kennt. Diese Dateien sind ausdrücklich abgeleitet und nicht offiziell. Wo die Feldtabellen von 1.1 von 2.1.0 abweichen, gilt die Tabelle aus 1.1.

Was wir ausdrücklich nicht tun

Wir verbessern den Standard nicht still. Jede Abweichung steht als `$comment` in der jeweiligen gebündelten Datei, trägt die Nummer des zugehörigen Fundes und ist im Werkzeug als Regel hinterlegt, nicht als Handänderung an einer Schemadatei.

Wir tragen keine Korrektur länger als nötig. Der Generator prüft vor jedem Lauf, ob der Defekt im Original überhaupt noch vorliegt. Ist er behoben, meldet er die Korrektur als überflüssig, statt sie weiter anzuwenden. Zwei Funde haben sich auf diesem Weg bereits erledigt: die Dateibenennung des Factsheets in 2.0 und die Feldstruktur derselben Datei, beide inzwischen im Branch-Kopf korrigiert.

Wir behaupten keine Zertifizierung. Der Analyzer liefert diagnostische Hinweise. Er erteilt weder eine Zertifizierung noch eine Freigabe.

Selbst nachprüfen

Die Originale und der Abgleich liegen im Produkt und lassen sich nachvollziehen:

```
python3 tools/fetch_spec.py --all      # Originale aus dem offiziellen Repo holen
python -m tools.sync_schemas          # melden, was abweicht und was überflüssig ist
python -m tools.spec_field_tables --diff # Felddaten gegen die Schemata halten
```

`VDA5050_Specs_Original/SOURCES.json` hält je Version Commit, Datum und die Liste der Dateien fest, die ein strikter Parser ablehnt. `app/validator/schemas/SOURCES.json` hält je gebündelter Datei Commit, Prüfsumme und die angewandten Korrekturen fest.

Rückmeldung

Wer hier einen Fehler findet, in unserer Auslegung oder in der Belegstelle, soll ihn uns nennen. Wir korrigieren das Dokument und, wenn nötig, das Produkt.

support@logiq-vda5050.com

VDA5050, M2X und LIF sind Marken der jeweiligen Gremien. LogIQ VDA5050 ist ein unabhängiges Werkzeug und von diesen weder autorisiert noch anerkannt.
